Attachment 8A

Model S371
DOS/InSpector
Programmer's Interface

User's Manual

V1.0 1/95
CISE 937

Canberra Industries, Inc., 800 Research Parkway, Meriden, CT 06450
Tel: (203) 238-2351 Telex: 643251 FAX: (203) 235-1347

The information in this manual describes the product as accurately as possible, but is subject to change without notice.

Printed in the United States of America.

Table of Contents

1. INTRODUCTION

This document for the Model S317 Inspector Programming Interface describes all functions of the InSpector's Applications Program Interface (API). All input and output interfaces of each module are provided in enough detail to allow programmers to code software for controlling the InSpector from a DOS or a Windows environment.

The C functions in the InSpector API software may be called from the Microsoft Assembly Language as well as from a number of high-level languages, including C, C++ and BASIC. The static DOS library included in the Model S371 package can be used only with Microsoft compilers such as:

Microsoft Visual Basic for DOS, V1.0
Microsoft C/C++, V7.0
Microsoft Visual C, V1.0, V1.5

The dynamic link library (DLL) can be used with any compiler capable of calling a function in a DLL written for Windows 3.1.

1.1. Nomenclature And Term Definitions

Standard C language programming conventions have been used throughout this document.

1.2. Prefixes and Data Types

The notation used in this document uses prefixes on symbol names to help the programmer determine data types and avoid errors before they reach the compiler.

Prefix	Definition	Data type	Quantity
b	BYTE	unsigned char	8 bits
s	SHORT	short or int	16 bits
i		int	16 bits
us	USHORT	unsigned short	16 bits
l	LONG	long	32 bits
ul	ULONG	unsigned long	32 bits
r	REAL	float	32 bits in IEEE FLOAT format
p		pointer	
en		enumerative	16 bits
st		structure	variable

1.3. Hexadecimal Notation

Descriptions of hexadecimal data items use the notation 'xxh', where 'xx' is the hexidecimal value.

2. HARDWARE DESIGN OVERVIEW

2.1. Communication Setup

Communication to and from the MCA board is via RS-232 at 1200, 9600, 19200, 38400, 57600, and 115200 baud, using 8 data bits, even parity, 1 stop bit. The MCA board will automatically adjust to the proper baud rate upon detecting serial data from the host. No hardware or software handshaking is used during communication.

Communication setup is initiated by the user application by using the InSpector API function sBaudSynch. The function is supplied with the proper baud rate.

2.2. Data Storage

Data acquired by the data acquisition hardware or downloaded into the MCA board via the **sLoadSpectrumData** command will be stored in channels 0 through 8191. During data acquisition channels 0 and 1 will hold elapsed live and true time respectively in 0.01 second resolution. Channels 2 through 8191 will hold spectral data. Each channel has 32 bits of storage capacity, or 4,294,967,295 counts. For the time channels this translates to approximately 11,930 hours. It is the responsibility of the user application to account for this information.

3. USER INTERFACE TO API

3.1. Command Summary

All functions will return with an integer code. This return code represents the status of the MCA, communication interface problems to the MCA, or parameter values that are out of range. A function that completes successfully will return with a value of 0 unless otherwise stated in the detailed description of the command.

Basic Commands

Other Supported Commands

3.2. Basic Commands

3.2.1. sAckStatus Command

short __far __pascal sAckStatus (InSpector_T * pst, SHORT s)

Resets status and alert flag bits in the MCASTATUS response. Other bits of other responses may be affected as noted. The power supply error code can also be cleared through this command.

pst - Pointer to an InSpector against which to apply the command.

s - Bitmap of which conditions in MCASTATUS to clear.

sResetPresetReached (0001h)	Resets the sPresetReached status flag in the MCASTATUS which indicates that the MCA has reached its preset time. Will not clear data or time.
sResetParmInit (0002h)	Resets the sMCAParmInit alert flag in the MCASTATUS which indicates that the MCA board has been re-initialized back to default values.
sResetAcquisitionDelay (0004h)	Resets sAcquisitionDelay status flag in MCASTATUS which indicates that the start of acquisition has been delayed. This command should not be used to terminate delayed acquisition. Use sAcquireOff command instead.
sResetRAMBatteryLow (0008h)	Resets sRAMBatLow alert flag in MCASTATUS which indicates that the Lithium battery for the ADC memory is low and the ADC memory is in danger of losing data.
sResetStabilizerOver (0010h)	Resets the sStabilizerOverrange alert flag in MCASTATUS
sResetPowerLost (0020h)	Resets the sPowerLost alert flag in MCASTATUS which indicates that power to the ADC/AMP hardware was temporarily interrupted during acquisition.
sResetHVStatus (0040h)	Resets the High Voltage Power Supply and updates the associated FAULT/INHIBIT status indications from the MCASTATUS response. Also clears the last power-supply error code. Refer to sGetHVStatus and MCASTATUS response to indicate which bits are affected by this command.
sResetBatteriesWarning (0100h)	Resets sBatInWarnAlert alert flag in MCASTATUS which indicates that the power supply batteries are both in the WARNING region. Refer to sGetPSStatus to indicate which bits are affected. The Quad-Enable Fault bit in the sGetPSStatus command will be reset.
sResetPSAlert (0200h)	Resets sLVAlert alert flag in MCASTATUS which indicates that the low voltage power supply (LVPS) has encountered any of the following conditions: a) Power source switch from battery A to B or from B to A occured. b) Interruption in ac power occurred. c) Both source batteries are in WARNING region; sBatInWarnAlert alert flag in MCASTATUS will also be set.

Use sGetPSStatus command to determine the actual cause, present operating battery, and battery-operating region. Refer to sGetPSStatus to indicate which bits are affected by this command. The Quad-Enable Fault bit in the sGetPSStatus command will be reset.

sResetPSCommError (0400h) — Reset sPSCommError alert flag in MCASTATUS, indicating that an error occurred during communication with the power supply. Use sGetPSStatus command to retrieve the last error code that may have triggered the alert. Refer to Appendix C for description of error codes. Refer to sGetPSStatus to indicate which bits are affected. The Quad-Enable Fault bit in the sGetPSStatus command will be reset.

The bitmap in s consists of the bitwise OR of the bits described. Bits other than the ones described will have no effect.

3.2.2. sAcquireOff Command

short __far __pascal sAcquireOff (InSpector_T * pst)

Turns acquisition logic OFF if presently ON.

pst - Pointer to an InSpector against which to apply the command.

This command will return an error if issued when acquisition is already off.

3.2.3. sAcquireStart Command

```
short __far __pascal sAcquireStart (    InSpector_T *    pst,
                                        SHORT            s,
                                        LONG             lDelay,
                                        ULONG            ulPresetTime,
                                        ULONG            ulLiveTime,
                                        ULONG            ulTrueTime )
```

Loads the amplifier, ADC, and stabilizer electronics with the parameters in the InSpector data structure. Turns on the high voltage power supply. Enables the data acquisition logic.

pst - Pointer to an InSpector against which to apply the command. Parameter data for the InSpector components must be completed before issuing the command. The fields that need to be set are:

- stADC

  enGainRange - Can be either:

enGain256	(00)
enGain512	(01)
enGain1k	(02)
enGain2k	(03)
enGain4k	(04)
enGain8k	(05)

  rLLD - LLD, value can be between 0.1 and 110.0

  rULD - ULD, value can be between 0.0 and 110.0

  rZero - ZERO, value can be between -5.0 and 5.0

enInput - Input source. Value can be:

enIntInputSource (01)
enExtInputSource (02)
enTestInputSource (03)

- stAMP

s - Bitmap that includes the flags:

sFastShaping (04h) - Shaping is Fast. If not set, shaping is slow.

sNegInhibitSignal (08h) - INHIBIT signal polarity is negative. If not set, signal polarity is positive

sPUROn (10h) - PUR is on. If not set, PUR is off.

sNegSignalPolarity (40h) - Negative signal polarity. If not set, polarity is positive.

sTRPPreamp (80h) - TRP Preamp type. If not set, it is an RC type preamp.

The bitmap in s consists of the bitwise OR of the bits described.

enCoarseGain - Coarse gain. The value can be:

enX2 (20h) - Gain of 2
enX6_6 (22h) - Gain of 6.6
enX10 (00h) - Gain of 10
enX33 (02h) - Gain of 33
enX120 (01h) - Gain of 120
enX400 (03h) - Gain of 400

rFineGain - Fine gain, value can be between 1.0 and 4.0

rSuperFineGain - Super Fine gain, value can be between 0.9800 and 1.0200

sPoleZero - Pole/zero Setting

- stHV

s - HVPS Status - bits encoded as follows:

sOn (01h) - HV ON/OFF status
0 = OFF,
1 = ON
Valid only if sInhibitChange is not set.

sHighRange (04h) - HV Range
0 = low,
1 = high

sInhibitChange (80h) - Inhibit change to high voltage state (ignore sOn)

The bitmap in s consists of the bitwise OR of the bits described.

r - Actual high voltage setting, including polarity

This DAC value will be retained by the MCA board and returned to the host as long as the high voltage is OFF. When the high voltage is turned ON then the actual value read from the HVPS will be returned instead.

- stStabilizer

stGain and stZero - Fields of each control structure are specified. The fields specified are:

sCentroid - Centroid location

sWindow - Width of window

sSpacing - Spacing of centroid

sCorrection - Correction

enControl - The control values can be:
enControlHold (00)
enControlOn (01)
enControlOff (02)
enControlLoad (03)

sRateDiv - Rate Divider, value can be 1, 2, 4, or 8

s - Stabilizer Control, bits encoded as follows:

sNaIRange (01h) - Stabilizer Gain Range
0 = ± 1% (for Ge detectors)
1 = ± 10% (for NaI detectors)

lDelay - Delay to start in seconds. 0 = no delay

ulPresetTime - Preset Time in 0.01 second increments

ulLiveTime - Elapsed Live Time in 0.01 second increments

ulTrueTime - Elapsed True Time in 0.01 second increments

If the sAcquireStart command is received while in PwrSAVE mode, the MCA will automatically switch the power mode to PwrON, start the acquisition, and return to PwrSAVE when acquisition terminates. The Delay-to-start value determines how long to delay the start of acquisition after the high voltage is stabilized. If the start of acquisition has been delayed, the Acquisition-Delay flag in the MCASTATUS will be set. This command will return an error if issued when acquisition is already on or the start of acquisition has been delayed.

Unlike the sAcquireON command this command performs a set of hardware commands that are necessary to prepare the hardware for acquisition. The values in the InSpector structure must be filled in completely for the command to perform properly. Unexpected results may occur if the values of the structure are not completed.

3.2.4. sBaudSynch Command

short __far __pascal sBaudSynch (InSpector_T * pst, Baud_T en)

The purpose of this command is to establish communication between a COMM port on the PC and the InSpector. After the COMM port is initialized, the hardware is sent a command repetitively to help it determine the operating baud rate. The MCA is allowed to cycle through the available baud rates and respond when the full command has been received error-free.

Since hardware and software configurations can affect the computer's communication speed, not all computer systems will be capable of operating at each baud rate. The use of the Windows DLL may also decrease the available baud rates.

pst - Pointer to an InSpector against which to apply the command. The value of the serial communication port must be completed.

en - The operating baud rate expressed as the divisor latch value to the UART. Its values can be:

B1200 (60h)
B9600 (0Ch)
B19200 (06h)
B38400 (03h)
B57600 (02h)
B115200 (01h)

3.2.5. sClearAcqData Command

```
short __far __pascal sClearAcqData (    InSpector_T *    pst,
                                        SHORT            s,
                                        SHORT            sFirst,
                                        SHORT            sLength )
```

Clears data and/or timers on MCA board

pst - Pointer to an InSpector against which to apply the command.

s - Bitmap used to clear time and/or data

sClearTime (40h) - Clears Time
sClearData (80h) - Clears Data

The bitmap in s consists of the bitwise OR of the bits described. Bits other than the ones described will have no effect.

sFirst - First channel to clear

sLength - Number of channels to clear.

Length must be 1 or greater, and sum of start plus length must not exceed 8192.

3.2.6. sGetADCParms Command

short __far __pascal sGetADCParms (InSpector_T * pst, ADC_T * pstADC)

Reads ADC parameters from the MCA board.

pst - Pointer to an InSpector against which to apply the command.

pstADC - Pointer to an ADC structure in which the parameter data is put. The fields that are affected are:

enGainRange - Will be one of the following:

enGain256 (00)
enGain512 (01)
enGain1k (02)

enGain2k (03)
enGain4k (04)
enGain8k (05)

rLLD - LLD, value will be between 0.1 and 110.0

rULD - ULD, value will be between 0.0 and 110.0

rZero - ZERO, value will be between -5.0 and 5.0

enInput - Input source. Value can be:

enIntInputSource (01)
enExtInputSource (02)
enTestInputSource (03)

3.2.7. sGetAMPParms Command

short __far __pascal sGetAMPParms (InSpector_T * pst, AMP_T * pstAMP)

Reads amplifier parameters from the MCA board.

pst - Pointer to an InSpector against which to apply the command.

pstAMP - Pointer to an AMP structure in which the parameter data is put. The fields that are affected are:

s - Bitmap that includes the flags:

sFastShaping (04h) - Shaping is Fast. If not set, shaping is slow.

sNegInhibitSignal (08h) - INHIBIT signal polarity is negative. If not set, signal polarity is positive.

sPUROn (10h) - PUR is on. If not set, PUR is off.

sNegSignalPolarity (40h) - Negative signal polarity. If not set, polarity is positive.

sTRPPreamp (80h) - TRP Preamp type. If not set, it is an RC preamp type.

The bitmap in s consists of the bitwise OR of the bits described.

enCoarseGain - Coarse gain. The value can be:

enX2 (20h) - Gain of 2
enX6_6 (22h) - Gain of 6.6
enX10 (00h) - Gain of 10
enX33 (02h) - Gain of 33
enX120 (01h) - Gain of 120
enX400 (03h) - Gain of 400

rFineGain - Fine gain, value can be between 1.0 and 4.0

rSuperFineGain - Super Fine gain, value can be between 0.9800 and 1.0200

sPoleZero - Pole/zero Setting

rShapingConst - Shaping constant code. The value is in microseconds. If the value is negative, then the constant cannot be determined by the InSpector API. Negative values can be:

rShapeOther 1 (-1) - First user-specific hardware shaping constant.
rShapeOther 2 (-2) - Second user-specific hardware shaping constant.

3.2.8. sGetHVStatus Command

short __far __pascal sGetHVStatus (InSpector_T * pst, HV_T * pstHV)

Reads High Voltage Power Supply status, DAC value, polarity, range, and other status information from high voltage hardware

pst - Pointer to an InSpector against which to apply the command.

pstHV - Pointer to a HV structure in which the parameter data is put. The fields that are affected are:

s - HVPS Status - bits encoded as follows:

sOn (01h) - HV ON/OFF status
0 = OFF
1 = ON
HV Status is OFF whenever the HVPS is in the FAULT or INHIBIT state

sHighRange (04h) - HV Range
0 = low,
1 = high

sLogicArmed (10h) - High voltage logic is ARMED if set. See MCASTATUS response.

sRamping (20h) - HV Ramping if set

sInhibited (40h) - HV Inhibited if set

sFault (80h) - HV Fault if set

NOTE: sInhibited and sFault are reset by the sAckHVStatus command, by the sAckStatus command (if sResetHVStatus set) and by the condition clearing. sRamping is reset by the condition clearing.

The bitmap in s consists of the bitwise OR of the bits described.

r - Actual high voltage reading, includes polarity

Note that r returns the actual voltage, not the voltage setting. The reading and setting should be equal if the high voltage is ON and not ramping. Ramping status is returned by the MCASTATUS response. When the high voltage power supply is OFF, the returned value is the same as that written with the sSetHVParms or sAcquireStart command. When the high voltage power supply is ON, the returned value will be the actual value read from the high voltage power supply.

3.2.9. sGetPSStatus Command

short __far __pascal sGetPSStatus (InSpector_T * pst, PS_T * pstPS)

Reads Power status and other power-supply related status information from MCA and Power-Supply boards.

pst - Pointer to an InSpector against which to apply the command.

pstPS - Pointer to a PS structure in which the parameter data is put. The fields that are affected are:

sSysStatus - Power Supply system status - bits encoded as follows:

sSwitched (01h)	- Power source switch occurred from battery A to B or B to A. This bit is reset by having the sResetPSAlert flag set in either the sAckPSStatus or sAckStatus command.
sInterrupt (02h)	- Interruption in ac power occurred. This bit is reset by having the sResetPSAlert flag set in either the sAckPSStatus or sAckStatus command.
sWarning (04h)	- Both source batteries are in the WARNING region. This bit is reset by having the sResetBatteriesWarning flag set in either the sAckStatus or sAckPSStatus command or by the condition ceasing to exist.
sCommError (10h)	- PS Communication error. sErrCode contains error code. This bit is reset by having the sResetPSCommError flag set in either the sAckStatus or sAckPSStatus command.
sQuadEnableFault (20h)	- Reset by issuing a sAckPSStatus command with or without acknowledge bits. Can also be cleared by issuing an sAckStatus command with sResetPSCommError, sResetBatteriesWarning, or sResetPSAlert set.

The bitmap in sSysStatus consists of the bitwise OR of the bits described.

enMode - Power mode of operation. The value can be:

enPwrSTDBY (01)	- Supply is in standby mode
enPwrSAVE (02)	- Supply is in power save mode
enPwrON (03)	- Supply is in power on

sPSStatus - Power Status - bits encoded as follows:

sBatBSupplyPower (04h)	- Power Source 0 = Battery A or ac power 1 = Battery B
sUseBatAFirst (10h)	- Power Source Mode 0 = Use the battery with the lowest charge first 1 = Use Battery A or ac power first
sACAvailable (80h)	- AC Power Available 0 = AC power is not available 1 = AC power is available and is connected to battery port A

stBatA and stBatB - Fields of each battery structure are affected. The fields affected are:

enOpRegion - Operating region, values can be:

enDischarged (00)
enStandby (01)

enBatteryLow (02)
enBatteryGood (03)

rVoltage - voltage, maximum value is 6.8V

sErrCode - Last Power-Supply Error code. Valid only when sCommError is set. Can be reset through sAckStatus or sAckPSStatus command. Refer to Appendix C for error code description.

sDelay - Standby Delay preset value in minutes

The Standby Delay value specified through the sSetPowerMode command indicates how long the MCA board will wait before switching the power supply to PwrSTDBY mode. The switch will be made only if the MCA board is not acquiring AND the communication between MCA and host computer stops. The elapsed delay will be reset when communication resumes. To resume communication properly, use the sBaudSynch command.

3.2.10. sGetPZStatus Command

short __far __pascal sGetPZStatus(InSpector_T * pst)

Reads pole/zero result.

pst - Pointer to an InSpector against which to apply the command.

The function returns with the pole/zero result. The value can be:

sNotPerformed (00h)	- Automatic pole/zero not performed
sSuccess (01h)	- Automatic pole/zero completed successfully
sPZTimeOut (FFh)	- Automatic pole/zero timed out error (not successful)

Any other value would be an InSpector API error code.

3.2.11. sGetStabilizerParms Command

short __far __pascal sGetStabilizerParms (InSpector_T * pst, Stabilizer_T * pstStab)

Reads Stabilizer parameters from the MCA board.

pst - Pointer to an InSpector against which to apply the command.

pstStab - Pointer to a Stabilizer structure in which the parameter data is put. The fields that are affected are:

stGain and stZero - Fields of each control structure are affected. The fields affected are:

sCentroid - Centroid location

sWindow - Width of window

sSpacing - Spacing of centroid

sCorrection - Correction

enControl - The control values can be:

enControlHold (00)

enControlOn (01)
enControlOff (02)
enControlLoad (03)

sRateDiv - Rate Divider, value can be 1, 2, 4, or 8

s - Status bits, encoded as follows:

sOverRange - OVERRANGE if set. Reset by ResetStabilizer command.

s - Stabilizer Control, bits encoded as follows:

sNalRange (01h) - Stabilizer Gain Range
0 = ± 1% (for Ge detectors)
1 = ± 10% (for Nal detectors)

OVERRANGE condition is set if Correction exceeds range (value at 000h or FFFh). Mode will be set to HOLD when OVERRANGE is set. OVERRANGE is reset by sResetStabilizer command.

3.2.12. sInitialize Command

short __far __pascal sInitialize (InSpector_T * pst, InitLevel_T en)

Forces MCA to re-initialize itself and reset data and variables as if power had been cycled. Actual level of initialization is specified on invocation of command.

pst - Pointer to an InSpector against which to apply the command.

en - Initialization level. Value can be:

enInitProgRAM (00) - Initialize program RAM only
enInitAllRAM (01) - Initialize program RAM and data RAM (time and data)

This command will reset the MCA's communication parameters. The sBaudSynch command should follow this command to properly re-establish the communication.

3.2.13. sLoadSpectrumData Command

```
short __far __pascal sLoadSpectrumData ( InSpector_T *    pst,
                                         SHORT            sFirst,
                                         SHORT            sLength,
                                         ULONG            ulTrueTime,
                                         ULONG            ulLiveTime,
                                         ULONG *          pul )
```

Writes raw spectral data into MCA board memory.

pst - Pointer to an InSpector against which to apply the command.

sFirst - Start channel

sLength - Length in channels

ulTrueTime - Spectrum Elapsed True Time in 0.01 second increments

ulLiveTime - Spectrum Elapsed Live Time in 0.01 second increments

pul - Pointer to an array of raw data

Length must be 1 or greater, and sum of start plus length must not exceed 8192. Command will not be honored if acquisition is on or start of acquisition has been delayed. In such a case, an error code is returned.

3.2.14. sPoleZero Command

```
short __far __pascal sPoleZero (   InSpector_T *   pst,
                                   SHORT           sInitStep,
                                   SHORT           s,
                                   SHORT           sTimeout,
                                   SHORT           sLLD,
                                   SHORT           sULD )
```

Directs MCA board to perform automatic pole zero adjustment on amplifier circuitry.

pst - Pointer to an InSpector against which to apply the command.

sInitStep - Initial step size to use to increment or decrement the hardware DAC, expressed as 2^n where n = 0 through 9 for 1 through 512. The initial value will be used until the hardware changes from overcompensation to undercompensation, or vice-versa. Thereafter a step size of 1 will be used.

s - Control byte, bits encoded as follows

sResetPZAlgorithm (80h) - Reset current pole/zero algorithm pattern if set. Otherwise, continue using existing pole/zero pattern

sWaitForCompletion (40h) - Command does not return until Pole Zero is complete. If not set, completion can be checked using the sGetPZStatus command.

sTimeout - Timeout value in seconds. This value determines how long to wait in the pole/zero algorithm attempting to perform pole/zero before resuming the normal program and servicing host commands. Valid range is

00	Use previously-established value in EEPROM
01-FEh	Value to use, in seconds. This value will be permanently written into EEPROM until (a) erased, or (b) new value is written.
FF	Erase previously-established value and use default which is 25 seconds

sLLD - Temporary LLD value for the ADC to use during the pole zero algorithm, expressed as a percentage. Valid range is

00	Use previously-established value in EEPROM
01-6Eh	Value to use, in percent, representing 1% to 110%. This value will be permanently written into EEPROM until (a) erased, or (b) new value is written.
FF	Erase previously-established value and use default which is 50 percent (32h).

sULD - Temporary ULD value for the ADC to use during the pole zero algorithm, expressed as a percentage. Valid range is

00	Use previously-established value in EEPROM
01-6Eh	Value to use, in percent, representing 1% to 110%. This value will be permanently written into EEPROM until (a) erased, or (b) new value is written.
FF	Erase previously-established value and use default which is 100 percent (64h).

If sWaitForCompletion is set, the function returns with the Pole/Zero result. The value can be:

sNotPerformed (00h)	- Automatic pole/zero not performed
sSuccess (01h)	- Automatic pole/zero completed successfully
sPZTimeOut (FFh)	- Automatic pole/zero timed out error (not successful)

Any other value would be an InSpector API error code.

If sWaitForCompletion is not set, pole/zero status can be checked via sGetPZStatus command.

If the application does not wait for the pole/zero to complete, the MCA board will NOT service host commands while in the pole zero algorithm. Commands sent will time out. The application can continuously poll the MCA board with a command, i.e. sGetPZStatus, until a valid response is received.

The internal pole/zero algorithm's pattern counters are reset:
(a) On system initialization following power on,
(b) On command (sResetPZAlgorithm) and
(c) Upon a successful pole/zero convergence.

3.2.15. sReadDisplayData Command

```
short __far __pascal sReadDisplayData (  InSpector_T *   pst,
                                         SHORT           sFirst,
                                         SHORT           sLength,
                                         SHORT           sK,
                                         SHORT           s,
                                         SHORT           sV,
                                         DspData_T *     pstDisplayData )
```

Returns spectral data from the MCA board scaled for display or plotting. Data scaling is based on the compression factor K, vertical full scale V, and number of data channels.

pst - Pointer to an InSpector against which to apply the command.

sFirst - Start channel

sLength - Length in channels

sK - Compression Factor (K) in channels. Either 1,2,4, or 8

s - Display Data control. Bits are encoded as follows:

sAutoscale (01h)	- Set autoscaling
sLog (02h)	- Set logarithmic scaling. Otherwise, it is linear.

sV - Vertical Full Scale (V), value is 2's exponent. Valid values are:

For Linear:

6	(VFS = 64)
7	(VFS = 128)
8	(VFS = 256)
•	
•	
•	
32	(VFS = 4.3KM)

For Log:

18	(VFS = 262K)
24	(VFS = 16M)
32	(VFS = 4.3KM)

pstDisplayData - Pointer to a structure to hold display data. The fields affected are:

s	- Similar in structure to the s that is passed. It only sets sLog if appropriate.
sV	- Similar in structure to the sV that is passed.
sLength	- Number of compressed channels that is returned.
sRawData	- The raw data from the command. Other bytes of raw data directly follow this field in the structure.

The buffer pointed to by pstDisplayData must be sufficiently large to include all the fields in DspData_T plus the additional display data that follows sRawData. Note that sSpareZ and sFill are not necessarily located at their respective positions in DspData_T. However, they are included so that space is allocated for them when creating a buffer.

Length must be 1 or greater, and sum of start plus length must not exceed 8192. Time data in channels 0 and 1 of the spectral data is excluded from the scaling.

3.2.16. sReadMCAInfo Command

short __far __pascal sReadMCAInfo (InSpector_T * pst, HardwareInfo_T * pstInformation)

Returns MCA firmware version, serial number, and model number, if any. The Version level is retrieved from ROM. The Model and Serial are retrieved from EEPROM.

pst - Pointer to an InSpector against which to apply the command.

pstInfo - Pointer to a hardware information structure in which the parameter data is put. The fields that are affected are:

lVersion	- MCA firmware version
lModel	- MCA Model
ulSerialNum	- MCA Serial Number

3.2.17. sReadMCAParms Command

short __far __pascal sReadMCAParms (InSpector_T * pst, MCAParms_T * pstParms)

Returns current MCA acquisition setup parameters (acquisition presets, start delay, etc.)

pst - Pointer to an InSpector against which to apply the command.

pstParms - Pointer to an MCA parameters structure in which the parameter data is put. The fields that are affected are:

s - Acquisition Preset Control.

sSetLiveMode (01h) - Preset mode is Live Time. If not set, the mode is true time.

sWaitAcquisition (02h) - Waiting for acquisition to start.

The bitmap in s consists of the bitwise OR of the bits described.

lDelay - Delay to start in seconds. 0 = no delay

ulPresetTime - Preset Time in 0.01 second increments

ulLiveTime - Elapsed Live Time in 0.01 second increments

ulTrueTime - Elapsed True Time in 0.01 second increments

When sWaitAcquisition is set it indicates that an acquire start command has been received but the hardware is not ready. The MCA board is presently waiting for the hardware to become ready. Meanwhile the elapsed delay since the acquire start command was issued is reported in the elapsed true-time field.

3.2.18. sReadMCAStatus Command

short __far __pascal sReadMCAStatus (InSpector_T * pst, MCAStatus_T * pstStatus)

Returns present MCA status

pst - Pointer to an InSpector against which to apply the command.

pstStatus - Pointer to an MCA Status structure in which the parameter data is put. The fields that are affected are:

sStatus - MCA Status, bits encoded as follows

sAcquisitionDelay (01h) - Acquisition-Delay flag, indicating that the MCA board has deferred the start of acquisition. Bit will reset itself when acquisition starts or by any of the following commands:

a) sAcquireOFF command, or
b) sAckMCAStatus or sAckStatus command with sResetAcquisitionDelay set

The elapsed delay time can be obtained through the sReadMCAParms command.

sMCACollecting (02h) - MCA Collecting flag, indicating the MCA board's data acquisition circuitry has been enabled. This bit resets when acquisition stops as result of having reached preset condition

or having been prematurely terminated via sAcquireOFF command.

sPresetReached (04h) - Preset-Reached flag, indicating that the MCA's data acquisition had reached its destined preset time and the data acquisition circuitry has been disabled. This bit is set by the MCA board during acquisition when the elapsed time equals or exceeds the preset time. The bit is reset by sAckStatus or sAckMCAStatus command with sResetPresetReached set.

sHVRamping (08h) - High voltage RAMPING. This is an indication that the high voltage has not yet reached its destined value. Bit will reset itself when condition ceases to exist.

sAlerts - Alert Flags. Based on which alert flag bit is set, the host may issue additional commands to find out more about the condition(s) and/or to reset the alerting status. The encoding of the bits follows. The condition is true when the bit is set (bit = 1).

sLVAlert (01h) - Low-Voltage-Power-Supply Alerting flag. Set by MCA board as it monitors the Power-Supply and finds any of these conditions true:

a) Power source switch has occurred from battery A to B or B to A
b) Interruption in AC power occurred
c) Both source batteries are in WARNING region (sBatInWarnAlert will also be set to 1)

Reset by sAckStatus or sAckPSStatus command with sResetPSAlert set

sHVAlert (02h) - High-Voltage-Power-Supply Alerting flag. Set by MCA board as it monitors the high voltage power supply and finds any of these conditions true:

a) High voltage output is INHIBITED
b) High voltage output is in FAULT state

Reset by sAckStatus or sAckHVStatus command or the condition(s) clearing.

sMCAParmInit (04h) - MCA Parameter-Initialization flag. Set by the MCA board to indicate that the program has re-initialized back to default values. Reset by sAckStatus or sAckMCAStatus command with sResetParmInit set.

sPSCommError (08h) - Power-Supply-Communication error flag. Set by the MCA board to inform the host computer that an error occurred during communication with the power supply. Use sGetPSStatus command to retrieve the last error code that may have set the flag. Refer to Appendix C for description of codes. Reset by sAckStatus or sAckPSStatus command with sResetPSCommError set.

sRAMBatLow (10h) - BB-RAM Low-Battery flag. Set by MCA board to indicate that the Lithium battery for the ADC memory needs replacing. Acquired data is in danger of being lost. Reset by sAckStatus or sAckMCAStatus command with

		sResetRAMBatteryLow set or if the condition ceases to exist.
	sStabilizerOverrange (20h)	- Stabilizer-Overrange alert flag indicates that the stabilizer is in the OVERRANGE condition. Reset by sAckStatus or sAckMCAStatus command with sResetStabilizerOver set.
	sBatInWarnAlert (40h)	- Batteries-in-Warning alert flag indicates that both power-supply batteries are operating in the WARNING region. Reset by sAckStatus or sAckPSStatus command with sResetBatteriesWarning set or when the condition ceases to exist. sLVAlert will also be set when this condition exists.
	sPowerLost (80h)	- Power-Lost alert flag indicates that power to the ADC, amplifier, and stabilizer electronics was lost during acquisition. Reset by sAckStatus or sAckMCAStatus command with sResetPowerLost set.
sStates	- States flag	
	sPwrSaveState (01h)	- Switch-to-PwrSave flag. This indicates that the MCA will automatically switch the power mode to PwrSAVE as soon as acquisition terminates. Refer to sAcquireStart, sAcquireOn and sSetPowerMode commands for additional information.
	sHVArmed (02h)	- High voltage is in the ARMED state, indicating that the voltage will ramp to the desired value when an sAcquireOn command is received.
sDiagStatus	- Diagnostic Status	

3.2.19. sReadPSInfo Command

```
short __far __pascal sReadPSInfo ( InSpector_T * pst, HardwareInfo_T * pstInfo )
```

Returns Power-Supply firmware version, serial number, and model number, if any. The Version level is retrieved from ROM. The Model and Serial number are retrieved from EEPROM.

pst - Pointer to an InSpector against which to apply the command.

pstInfo - Pointer to a hardware information structure in which the parameter data is put. The fields that are affected are:

lVersion	- PS firmware version
lModel	- PS Model
ulSerialNum	- PS Serial Number

3.2.20. sReadROI Command

```
short __far __pascal sReadROI (    InSpector_T *   pst,
                                   SHORT           sFirst,
```

```
                                        SHORT       sLength,
                                        ULONG *     pul )
```

Returns ROI total from MCA board

pst - Pointer to an InSpector against which to apply the command.

sFirst - Start channel

sLength - Length in channels

pul - pointer to a long integer to place the ROI total.

Length must be 1 or greater, and sum of start plus length must not exceed 8192.

3.2.21. sReadSpectrumData Command

```
short __far __pascal sReadSpectrumData ( InSpector_T *    pst,
                                         SHORT            sFirst,
                                         SHORT            sLength,
                                         ULONG *          pul )
```

Returns spectral data from MCA board.

pst - Pointer to an InSpector against which to apply the command.

sFirst - Start channel

sLength - Length in channels

pul - pointer to a buffer to place the spectral data

Length must be 1 or greater, and sum of start plus length must not exceed 8192.

3.2.22. sResetStabilizer Command

```
short __far __pascal sResetStabilizer (    InSpector_T * pst )
```

Resets OVERRANGE status on stabilizer circuitry and loads Correction DACs to default values. Also resets Stabilizer-Overrange alert flag in MCASTATUS if set.

pst - Pointer to an InSpector against which to apply the command.

3.2.23. sSetPowerMode Command

```
short __far __pascal sSetPowerMode (    InSpector_T *    pst,
                                         PowerMode_T      enPowerMode,
                                         SHORT            s,
                                         SHORT            sDelay )
```

Instructs MCA to change the operating power supply mode.

pst - Pointer to an InSpector against which to apply the command.

enPowerMode - Power mode of operation (mX). The value can be:

enPwrOFF (00)	- Set supply to off
enPwrSTDBY (01)	- Set supply to standby mode
enPwrSAVE (02)	- Set supply to power save mode
enPwrON (03)	- Set supply to power on

s - Control - bits encoded as follows:

sUseBatAFirst (10h) - Power Source Mode (sX)
0 = Use battery with lowest charge first
1 = Use Battery A/AC first

sInhibitPowerSource (20h) - Inhibit change to power source sX with this command
0 = allow change to sX
1 = do not allow change to sX

sInhibitPowerMode (40h) - Inhibit change to power mode mX with this command
0 = allow change to mX
1 = do not allow change to mX

sDelay - Delay to switch power mode to PwrSTDBY in minutes if system idling (no communication activity between host and MCA). Value 0000 = inhibit automatic switch to PwrSTDBY if system is idling. Maximum time allowed is 18 hours.

In OFF no power is applied to the electronics. Cycling the main power switch from OFF to ON switches the power supply to PwrStdby mode. In PwrStdby mode, minimum electronics are ON. The electronics will monitor host communication and upon detection will automatically switch the system to PwrSave mode in which only the power supply and MCA electronics are ON. No power is applied to the data acquisition or high voltage electronics in PwrSave mode. In PwrOn mode, power is applied to the MCA and data-acquisition electronics. The host computer must instruct the MCA to switch power modes from PwrSave to PwrOn. High voltage electronics can only be enabled when the system is in PwrON mode.

While acquisition is ON and a command is received from the host computer to switch from PwrOn to PwrSave the command will be carried out when acquisition terminates. The state of the Switch-to-PwrSave flag can be read back via MCASTATUS.

3.2.24. sSetPwrMode

short __far __pascal sSetPwrMode (InSpector_T * pst, PowerMode_T enPowerMode)

This command performs only the power mode setting of the sSetPowerMode command. It calls the sSetPowerMode command with the Power Source Inhibited flag set.

3.2.25. sSetPwrSource

short __far __pascal sSetPwrSource (InSpector_T * pst, SHORT s)

This command performs only the power source setting of the sSetPowerMode command. It calls the sSetPowerMode command with the Power Mode Inhibited flag set.

3.2.26. sSetStandbyDelay

short __far __pascal sSetStandbyDelay (InSpector_T * pst, SHORT s)

This command performs only the delay setting of the sSetPowerMode command. It calls the sSetPowerMode command with the Power Mode and Power Source Inhibited flags set.

3.3. Other Supported Commands

3.3.1. sAckHVStatus Command

short __far __pascal sAckHVStatus (InSpector_T * pst)

Resets the High Voltage Power Supply and updates the associated FAULT/INHIBITED status indications from the MCASTATUS response. Also clears the last power-supply error code.

pst - Pointer to an InSpector against which to apply the command.

Refer to sGetHVStatus and MCASTATUS response to indicate which bits are affected by this command.

3.3.2. sAckMCAStatus Command

short __far __pascal sAckMCAStatus (InSpector_T * pst, SHORT s)

Resets MCA status and alert flag bits in the MCASTATUS response. These include Preset-Reached, battery-low conditions, etc.

pst - Pointer to an InSpector against which to apply the command.

s - Bitmap of which conditions in MCASTATUS to clear.

sResetPresetReached (0001h)	Resets the sPresetReached status flag in the MCASTATUS which indicates that the MCA has reached its destined preset time. Will not clear data or time.
sResetParmInit (0002h)	Resets the sMCAParmInit alert flag in the MCASTATUS which indicates that the MCA board has been re-initialized back to default values.
sResetAcquisitionDelay (0004h)	Resets the sAcquisitionDelay status flag in MCASTATUS which indicates that the start of acquisition has been delayed. This command should not be used to terminate delayed acquisition. Use sAcquireOff command instead.
sResetRAMBatteryLow (0008h)	Resets the sRAMBatLow alert flag in MCASTATUS which indicates that the Lithium battery for the ADC memory is low and the ADC memory is in danger of losing data.
sResetStabilizerOver (0010h)	Resets the sStabilizerOverrange alert flag in MCASTATUS
sResetPowerLost (0020h)	Resets the sPowerLost alert flag in MCASTATUS which indicates that power to the ADC/AMP hardware was temporarily interrupted during acquisition.

3.3.3. sAckPSStatus Command

short __far __pascal sAckPSStatus (InSpector_T * pst, SHORT s)

Resets the Power Supply alert flag bits in MCASTATUS and Quad-Enable Fault bit in the sGetPSStatus command

pst - Pointer to an InSpector against which to apply the command.

s - Bitmap of which conditions in MCASTATUS to clear.

sResetBatteriesWarning (0100h)	Resets the sBatInWarn alert flag in MCASTATUS which indicates that the power supply batteries are both in the WARNING region.
sResetPSAlert (0200h)	Resets the sLVAlert alert flag in MCASTATUS which indicates that the low voltage power supply has encountered any of the following conditions: a) Power source switch from battery A to B or B to A occurred. b) Interruption in ac power occurred. c) Both source batteries are in WARNING region; sBatInWarn alert flag in MCASTATUS will also be set. Use sGetPSStatus command to determine the actual cause, present operating battery, and battery-operating region.
sResetPSCommError (0400h)	Reset sPSCommError alert flag in MCASTATUS) which indicates that an error occurred during communication with the power supply. Use sGetPSStatus command to retrieve the last error code that may have triggered the alert. Refer to Appendix C for description of error codes.

The bitmap in s consists of the bitwise OR of the bits described. Bits other than the ones described will have no effect. Refer to sGetPSStatus to indicate which bits are affected by this command. The Quad-Enable Fault bit in the sGetPSStatus command will be reset whenever the sAckPSStatus command is issued regardless of the bitmap.

3.3.4. sAcquireOn Command

```
short __far __pascal sAcquireOn (  InSpector_T *    pst,
                                   SHORT            s,
                                   LONG             lDelay,
                                   ULONG            ulPresetTime,
                                   ULONG            ulLiveTime,
                                   ULONG            ulTrueTime )
```

Reloads the amplifier, ADC, and stabilizer electronics with stored parameters and enables the data acquisition logic.

pst - Pointer to an InSpector against which to apply the command.

s - Acquisition Preset Control. Used to set the preset mode.

sSetLiveMode (01h) - Preset mode is Live Time. If not set, the mode is true time.

The control in s consists of the bitwise OR of the bits described. Bits other than the ones described will have no effect.

lDelay - Delay to start in seconds. 0 = no delay

ulPresetTime - Preset Time in 0.01 second increments

ulLiveTime - Elapsed Live Time in 0.01 second increments

ulTrueTime - Elapsed True Time in 0.01 second increments

If the sAcquireOn command is received while in PwrSave mode, the MCA will automatically switch the power mode to PwrON, start the acquisition, and return to PwrSave when acquisition terminates. The high voltage will not be automatically turned ON however unless the high voltage logic has been ARMED. Refer to the sSetHVParms command for description of the ARMED state. The present state of the ARMED and the Switch-to-PwrSave flags can be read back via MCASTATUS. The Delay-to-start value determines how long to delay the start of acquisition. If the start of acquisition has been delayed,

the Acquisition-Delay flag in the MCASTATUS will be set. An appropriate delay value should always be specified when starting acquisition with the high voltage in the ARMED state to allow for high voltage stabilization. This command will return an error if issued when acquisition is already on or the start of acquisition has been delayed.

3.3.5. sReadRawSpectralData Command

```
short __far __pascal sReadRawSpectralData (  InSpector_T *   pst,
                                             SHORT           sFirst,
                                             SHORT           sLength,
                                             ULONG *         pul )
```

Returns raw spectral data from MCA board in un-compressed form.

pst - Pointer to an InSpector against which to apply the command.

sFirst - Start channel

sLength - Length in channels

pul - pointer to a buffer to place the spectral data

Length must be 1 or greater, and sum of start plus length must not exceed 8192.

3.3.6. sSetAcquisitionDelay Command

short __far __pascal sSetAcquisitionDelay (InSpector_T * pst, LONG lDelay)

Sets the delay-to-start acquisition independently of the sAcquireOn or sAcquireStart command.

pst - Pointer to an InSpector against which to apply the command.

lDelay - Delay to start in second. 0000 = no delay

This is actually the same value that is set through the sAcquireOn command. It can be overwritten with a new value at any time with the sAcquireOn command. The present delay value can be read back through the sReadMCAParms command.

3.3.7. sSetADCParms Command

short __far __pascal sSetADCParms (InSpector_T * pst)

Sets up ADC parameters on the MCA board.

pst - Pointer to an InSpector against which to apply the command. Parameter data must be placed into the stADC structure. The fields that need to be set are:

enGainRange - Can be:

enGain256	(00)
enGain512	(01)
enGain1k	(02)
enGain2k	(03)
enGain4k	(04)
enGain8k	(05)

rLLD - LLD, value can be between 0.1 and 110.0

rULD - ULD, value can be between 0.0 and 110.0

rZero - ZERO, value can be between -5.0 and 5.0

enInput - Input source. Value can be:

enIntInputSource (01)
enExtInputSource (02)
enTestInputSource (03)

The ZERO DAC value specified in rZero will be modified within the MCA board according to the Zero Correction Table value for the specified Conversion Gain. This table has been preset on the hardware.

3.3.8. sSetAMPParms Command

short __far __pascal sSetAMPParms (InSpector_T * pst)

Sets up amplifier parameters on the MCA board.

pst - Pointer to an InSpector against which to apply the command. Parameter data must be placed into the stAMP structure. The fields that need to be specified are:

s - Bitmap that includes the flags:

sFastShaping (04h)	- Shaping is Fast. If not set, shaping is slow.
sNegInhibitSignal (08h)	- INHIBIT signal polarity is negative. If not set, signal polarity is positive
sPUROn (10h)	- PUR is on. If not set, PUR is off.
sNegSignalPolarity (40h)	- Negative signal polarity. If not set, polarity is positive.
sTRPPreamp (80h)	- TRP Preamp type. If not set, it is an RC preamp type.

The bitmap in s consists of the bitwise OR of the bits described.

enCoarseGain - Coarse gain. The value can be:

enX2 (20h)	- Gain of 2
enX6_6 (22h)	- Gain of 6.6
enX10 (00h)	- Gain of 10
enX33 (02h)	- Gain of 33
enX120 (01h)	- Gain of 120
enX400 (03h)	- Gain of 400

rFineGain - Fine gain, value can be between 1.0 and 4.0

rSuperFineGain - Super Fine gain, value can be between 0.9800 and 1.0200

sPoleZero - Pole/zero Setting

3.3.9. sSetHVParms Command

short __far __pascal sSetHVParms (InSpector_T * pst)

Sets HV voltage setting, polarity, range, and status on MCA board.

pst - Pointer to an InSpector against which to apply the command. Parameter data must be placed into the stHV structure. The fields that need to be specified are:

s - HVPS Status - bits encoded as follows:

sOn (01h)	- HV ON/OFF status 0 = OFF 1 = ON Valid only if sInhibitChange is not set.
sHighRange (04h)	- HV Range 0 = low 1 = high
sInhibitChange (80h)	- Inhibit change to high voltage state (ignore sOn)

The bitmap in s consists of the bitwise OR of the bits described.

r - Actual high voltage setting, including polarity

This value will be retained by the MCA board and returned to the host as long as the high voltage is OFF. When the high voltage is turned ON then the actual value read from the HVPS will be returned instead.

High voltage cannot be turned on if present power mode is not PwrOn. If this command is issued while in PwrSave mode the high voltage logic on the MCA board will be placed in the ARMED state. While in the ARMED state the high voltage will be automatically turned ON when an sAcquireOn command is received and automatically turned OFF when the acquisition terminates. The sAcquireOn command switches the power mode to ON to start acquisition and back to SAVE when done. The ARMED condition will be reset if the power mode is explicitly set to PwrOn before issuing the sAcquireOn command. With the high voltage in the ARMED state, an appropriate delay value should be specified when issuing the sAcquireOn command to force a wait before actually starting acquisition to allow for high voltage stabilization. Change of polarity and range will not be allowed if high voltage is already on.

3.3.10. sSetStabilizerParms Command

short __far __pascal sSetStabilizerParms (InSpector_T * pst)

Sets up stabilizer parameters on the MCA board.

pst - Pointer to an InSpector against which to apply the command. Parameter data must be placed into the stStab structure. The fields that need to be specified are:

stGain and stZero - Fields of each control structure are specified. The fields specified are:

sCentroid	- Centroid location
sWindow	- Width of window
sSpacing	- Spacing of centroid
sCorrection	- Correction

enControl - The control values can be:

enControlHold (00)
enControlOn (01)
enControlOff (02)
enControlLoad (03)

sRateDiv - Rate Divider, value can be 1, 2, 4, or 8

s - Stabilizer Control, bits encoded as follows:

sNalRange (01h) - Stabilizer Gain Range
0 = ± 1% (for Ge detectors)
1 = ± 10% (for Nal detectors)

Zero and Gain Correction values will not be written to the hardware if the present stabilizer mode is ON.

3.4. Data Structures

3.4.1. ADC

```
typedef struct ADC_S {      GainRange_T   enGainRange;   /* Gain Range */
                            REAL          rLLD;          /* LLD */
                            REAL          rULD;          /* ULD */
                            REAL          rZero;         /* Zero */
                            Input_T       enInput;       /* ADC Input Source */
} ADC_T;
```

3.4.2. Amplifier

```
typedef struct AMP_S {      SHORT         s;             /* Parameters in b0 */
                            CoarseGain_T  enCoarseGain;  /* Coarse Gain */
                            REAL          rFineGain;     /* Fine Gain */
                            REAL          rSuperFineGain; /* Super Fine Gain */
                            SHORT         sPoleZero;     /* Power Zero setting */
                            REAL          rShapeConst;   /* Shaping Time Constant */
} AMP_T;
```

3.4.3. Battery

```
typedef struct Battery_s {  OpRegion_T    enOpRegion;    /* Operating region */
                            REAL          rVoltage;      /* Voltage */
} Battery_T;
```

3.4.4. Baud Rate

```
typedef enum Baud_E { B1200, B9600, B19200, B38400, B57600,B115200 } Baud_T;
                      =0x60, =0x0C, =6,     =3,     =2,     =1
```

3.4.5. Coarse Gain

```
typedef enum CoarseGain_E { enX2,   enX6.6, enX10, enX33, enX120,enX400 } CoarseGain_T;
                            =0x20, =0x22, =0x0,  =0x2,  =0x1,  =0x3
```

3.4.6. COMM Port

```
typedef enum CommPort_E { enCOM1,enCOM2 } CommPort_T;
                          =0x400, =0x402
```

3.4.7. Control Mode

```
typedef enum ControlMode_E   { enControlHold=0, enControlOn=1, enControlOff=2, enControlLoad=3
                             } ControlMode_T;
```

3.4.8. Control Structure

```
typedef struct Control_S {     SHORT          sCentroid;     /* Centroid */
                               SHORT          sSpacing;      /* Spacing */
                               SHORT          sCorrect       /* Correction */
                               ControlMode_T  enControl;     /* Control Mode */
                               SHORT          sRateDiv;      /* Rate Division */
                               SHORT          sWindow;       /* Window */
                               SHORT          s;             /* Overrange bit */
} Control_T;
```

3.4.9. Display Data

```
typedef struct DspData_T {     SHORT          sSpare1;       /* Spare */
                               SHORT          s;             /* Linear or Logarithmic scale */
                               SHORT          sV;            /* Vertical full scale - value is 2's exponent */
                               SHORT          sLength;       /* Length of raw data */
                               SHORT          sRawData;      /* Raw data for display */
                               SHORT          sSpare 2;      /* Spare */
                               SHORT          sFill[2];      /* Filler bytes */
} DspData_T;
```

3.4.10. Gain Range

```
typedef enum GainRange_E     { enGain256=0, enGain572=1, enGain1k=2, enGain2k=3, enGain4k=4, enGain8k=5
                             } GainRange_T;
```

3.4.11. Hardware Information

```
typedef struct HardwareInfo_S { LONG          lVersion;      /* Firmware Version */
                                LONG          lModel;        /* Power Supply Model */
                                ULONG         ulSerialNum;   /* Power Supply Serial Number */
} HardwareInfo_T;
```

3.4.12. High Voltage Power Supply

```
typedef struct HV_S {          SHORT          s;             /* HVPS Status */
                               REAL           r;             /* Voltage */
} HV_T;
```

3.4.13. Initialization Level

```
typedef enum InitLevel_E     { enInitProgRAM=0, enInitAllRAM=1
                             } InitLevel_T;
```

3.4.14. Input Source

```
typedef enum Input_E        { enIntInputSource=1, enExtInputSource=2, enTestInputSource=3
                            } Input_T;
```

3.4.15. InSpector

```
typedef struct InSpector_S {    CommPort_T      enCommPort;     /* COMM Port */
                                ADC_T           stADC;          /* ADC Information */
                                AMP_T           stAMP;          /* AMP Information */
                                HV_T            stHV;           /* HV Information */
                                Stabilizer_T    stStabilizer;   /* Stabilizer Information */
} InSpector_T;
```

3.4.16. MCA Parameters

```
typedef struct MCAParms_S {     SHORT           s;              /* Acquisition Preset Control */
                                LONG            lDelay;         /* Delay to start in seconds */
                                ULONG           ulPresetTime;   /* Preset Time */
                                ULONG           ulLiveTime;     /* Elapsed Live Time */
                                ULONG           ulTrueTime;     /* Elapsed True Time */
} MCAParms_T;
```

3.4.17. MCA Status

```
typedef struct MCAStatus_T {    SHORT           sStatus;        /* MCA Status */
                                SHORT           sAlerts;        /* Alert flags */
                                SHORT           sStates;        /* States flag */
                                SHORT           sDiagStatus;    /* Diagnostic Status */
} MCAStatus_T;
```

3.4.18. Operating Region

```
typedef enum OpRegion_E     { enDischarged=0, enStandby=1, enBatteryLow=2, enBatteryGood=3
                            } OpRegion_T;
```

3.4.19. Power Mode

```
typedef enum PowerMode_E    { enPwrOFF=0, enPwrSTDBY=1, enPwrSAVE=2, enPwrON=3
                            } PowerMode_T;
```

3.4.20. Power Supply

```
typedef struct PS_S         { SHORT         sSysStatus;     /* Power Supply System Status */
                              PowerMode_T   enMode;         /* Power Mode of operation */
                              SHORT         sPSStatus;      /* Power Status */
                              Battery_T     stBatA;         /* Battery A information */
                              Battery_T     stBatB;         /* Battery B information */
                              SHORT         sErrCode;       /* Last Power Supply Error code */
                              SHORT         sDelay;         /* Standby delay in minutes */
} PS_T;
```

3.4.21. Stabilizer

```
typedef struct Stabilizer_S {   Control_T       stGain;         /* Gain structure */
                                Control_T       stZero;         /* Zero structure */
```

```
                        SHORT       s;              /* Stabilizer Control bit */
} Stabilizer_T;
```

3.5. User Interface Example of Acquiring Data

```
        #include <dosinsp.h>

        InSpector_T MyInSpector;
        MCAStatus_T MyMCAStatus;
        int rc;
        ULONG MyData[4000];

/* Set communication port */

        MyInSpector.enCommPort = enCOM1;

/* Synch up with InSpector*/

        sBaudSynch(&MyInSpector,B19200);

/* Set analog-to-digital converter parameters */

        MyInSpector.stADC.enGainRange = enGain2k;
        MyInSpector.stADC.rLLD = 1.01;
        MyInSpector.stADC.rULD = 99.99;
        MyInSpector.stADC.rZero = 0.001;
        MyInSpector.stADC.enInput = enIntInputSource;

/* Set amplifier parameters */

        MyInSpector.stAMP.enCoarseGain = enX10;
        MyInSpector.stAMP.rFineGain = 1.56;
        MyInSpector.stAMP.rSuperFineGain = 1.00984;
        MyInSpector.stAMP.sPoleZero = 1000;
        MyInSpector.stAMP.s = sPUROn | sFastShaping;

/* Set high voltage */

        MyInSpector.stHV.s = sHighRange;
        MyInSpector.stHV.r = -1300;

/* Set stabilizer gain parameters */

        MyInSpector.stStabilizer.stGain.sCentroid = 200;
        MyInSpector.stStabilizer.stGain.sWindow = 5;
        MyInSpector.stStabilizer.stGain.sSpacing = 4;
        MyInSpector.stStabilizer.stGain.enControl = enControlOn;
        MyInSpector.stStabilizer.stGain.sRateDiv = 1;
        MyInSpector.stStabilizer.stGain.sCorrect = 0;

/* Set stabilizer zero parameters */

        MyInSpector.stStabilizer.stZero.sCentroid = 100;
        MyInSpector.stStabilizer.stZero.sWindow = 5;
        MyInSpector.stStabilizer.stZero.sSpacing = 4;
        MyInSpector.stStabilizer.stZero.enControl = enControlHold;
```

```
        MyInSpector.stStabilizer.stZero.sbcRateDiv = 1;
        MyInSpector.stStabilizer.stZero.sCorrect = 0;

/* Start Acquisition */

        rc = sAcquireStart( &MyInSpector, sSetLiveMode, 60, 600, 0, 0 );

/* Wait for preset condition to be reached */

        if (rc = =0) {
                do {
                                rc = sReadMCAStatus( &MyInSpector, &MyMCAStatus);
                } while ((rc = =0) && (MyMCAStatus.sStatus & sPresetReached) );

/* Read Spectrum Data */

                if (rc = =0) {
                                rc = sReadSpectrumData( &MyInSpector, 1, 4000, MyData );

                                if (rcl =0) {
                                                Perform error routine
                                }
                } else {
                                Perform error routine
                }
        } else {
                Perform error routine
        }
```

4. CALLING INSPECTOR API C FUNCTIONS

This section deals with calling the InSpector API C functions from BASIC and from C itself.

The method for calling C functions from another language described in this document was chosen primarily for its clarity and should not be taken as being the only method available to the programmer.

There are several major differences between BASIC and C which often prevent direct calling between the two languages.

a. The order in which parameters are pushed on the stack. BASIC parameters are pushed on the stack in the order in which they appear, whereas C parameters are normally pushed in reverse order to allow for varying number of arguments. However, the functions for the InSpector API use the BASIC calling convention. This convention is noted by using the __pascal keyword before the function prototypes in the description of the commands.

b. Formal arguments to subprograms in C are ordinarily passed by value, whereas in BASIC they are passed by reference.

c. By default, the C compiler inserts an underscore character (_) at the beginning of each name in the C module, whereas the BASIC compiler does not.

4.1. Calling Microsoft C Functions From BASIC

BASIC provides the DECLARE statement which provides the necessary interfacing for the calling of C functions and subroutines. The DECLARE statement tells the program that the subroutine or function is defined in an external file. It also defines the formal argument types of the function.

The CDECL attribute is not necessary for using the InSpector API's C functions because the C functions were compiled using the Pascal calling convention. BASIC uses this convention.

In addition, the functions VARPTR and VARSEG provided by BASIC allow for proper passing of array pointers to the C subprograms.

C requires that most of the parameters for its subroutines and functions are passed by value. True value passing is achieved in BASIC by applying BYVAL to the parameters in the DECLARE statement when declaring the external C subprograms.

When a C function requires a reference to a variable, a pointer variable is required. All pointers used in the InSpector API interface are far pointers. A far pointer to a variable is achieved in BASIC by applying SEG to the parameter in the DECLARE statement when declaring the external C subprogram.

Syntax for the interface declaration to call an InSpector API function which requires a reference and a value is as follows:

```
DECLARE FUNCTION fnname [LIB "WININSP"] (SEG arg1 as type, BYVAL arg2 as INTEGER, etc.)
```

where:

fnname name and type of function to be called from Visual BASIC for DOS

arg1..n formal arguments, including type

fnname includes both the name and type of function. Types are specified in BASIC by appending a type-declaration suffix to the name. Type-declaration suffixes are as follows:

%	16-bits	integer or short
&	32-bits	long
!	32-bits	single precision floating point
#	64-bits	double precision floating point
$	variable	strings

The LIB keyword and its value "WININSP" are required only when using the Windows DLL.

For example:

```
DECLARE FUNCTION sBaudSynch%
(SEG pst as InSpector_T, BYVAL en as INTEGER)
```

Note that the C function 'sBaudSynch' has been declared as returning a 16-bit quantity. The function requires two arguments, a pointer to an InSpector structure and the baud rate which is passed by value.

The BASIC functions VARPTR and VARSEG are used in the main program to produce offsets and segments, respectively, for arrays of values. When passing a pointer to an array in the InSpector API, the segment of the array is placed in the parameter list before the offset. The offset and segment are passed by value as integers.

All functions in the InSpector API return 16-bit quantities.

There are a few uses in the InSpector API of unsigned long integers. Since BASIC does not support unsigned integers, a signed long integer can be substituted. The memory contents remain unchanged when using a signed instead of an unsigned integer. However, the BASIC programmer must account for the differences in interpretation of signed and unsigned integers.

4.1.1. Compiling Programs Written in BASIC

Requirement for DOS:
Microsoft Visual BASIC for DOS, V1.0

Using the Microsoft Visual BASIC for DOS Compiler, a BASIC program named DEMOBAS.BAS can be compiled as follows:

```
bc DEMOBAS.BAS /O;
```

where:

bc	is the command to invoke the Visual BASIC for DOS Compiler
/O	creates object (.OBJ) file only
DEMOBAS.BAS	is the Visual BASIC for DOS program being compiled

The compiliation for BASIC Windows programs is similar.

4.1.2. Linking Programs Written in Visual BASIC for DOS

The linker must be the Microsoft Segmented Executable Linker V5.30, or later.

The syntax for linking DEMOBAS.BAS demonstration program is:

```
link /NOE /NOD DEMOBAS,,,basic.lib DOSINSP.LIB CRTINSP.LIB
```

where:

link	is the command to invoke the linker
/NOE, /NOD	are required
DEMOBAS	is the Visual BASIC for DOS demonstration program.
'basic.lib '	is a BASIC run-time library. Replace 'basic.lib' with the name of the appropriate library.
DOSINSP.LIB	is the InSpector API static library
CRTINSP.LIB	is the library containing needed C run-time routines.

Note that the order of the libraries is important.

4.1.3. Linking Programs Written in Visual BASIC for Windows

The syntax for linking a BASIC Windows program is:

```
link  basprog,,,basic.lib WININSP.LIB, basprog.def
```

where:

link	is the command to invoke the linker
'basprog'	is the BASIC Windows object file
'basic.lib'	is the BASIC runtime library. Replace 'basic.lib' with the name of the appropriate library
WININSP.LIB	is the InSpector API import library

'basprog.def' is a user-supplied module definition file

Note that to run the BASIC program, the file WININSP.DLL must be in a directory supplied in the PATH variable or must be in the current directory.

4.2. Calling C Functions From C

4.2.1. Compiling Microsoft C Functions for DOS

Requirements:
Microsoft C/C++ V7.0 or Visual C V1.0 or V1.5.

A C module can be compiled as follows:

```
cl -c -Ze -Alfu DEMOC.C
```

where:

cl	is the command to invoke Microsoft C Compiler
-c	no link step desired
-Ze	enables language extension keywords (FAR, etc.)
-Alfu	is the program configuration l = large code pointer f = Far data pointer u = SS not equal to DS; DS loaded for each module
DEMOC.C	is the C module being compiled

Note that the selected Program configuration must be the same for all modules.

4.2.2. Linking Microsoft C Programs for DOS

Requirement:
Microsoft Segmented Executable Linker V5.30, or later.

The syntax for linking the DEMOC.C program is:

```
link /NOE DEMOC,,,DOSINSP.LIB;
```

where:

link	is the command to invoke the linker
DEMOC	is the Microsoft C program
/NOE	is required
DOSINSP.LIB	is the InSpector API static library.

4.2.3. Compiling C Functions for Windows

A C module for Windows can be compiled as follows:

```
cl -c -Ze -GA -Gs cprog.c
```

where:

cl	is the command to invoke the compiler
-c	no link step desired
-Ze	enables language extension keywords (FAR, etc.)
-Ga	optimizes Entry/Exit code for Protected Mode Windows
-Gs	turns off stack checking
'cprog.c'	is the C module being compiled

4.2.4. Linking C Programs for Windows

The syntax for linking a C Windows program is:

```
link /NOD:SLIBCE cprog,,,WININSP.LIB LIBW.LIB SLIBCEW, cprog.def
```

where:

link	is the command which invokes the linker
/NOD:SLIBCE	excludes SLIBCE from the default library search. Replace SLIBCE with another library named if the default small library was not used in compiling the object file
'cprog'	is the C Windows object file
WININSP.LIB	is the InSpector API import library
SLIBCEW	is the small model application library for Windows; it may be replaced with another model library
LIBW.LIB	is import information for Windows DLLs
'cprog.def'	is a user-supplied module definition file

Note that to run the program, the file WININSP.DLL must be in a directory supplied in the PATH variable or must be in the current directory.

A. COMMUNICATION

A.1. Serial Communication Parameters

Data will be transmitted via RS-232 as follows:

Baud rate:	1200, 9600, 19200, 38400, 57600, or 115200 bps
Parity:	Even
Word Length:	8 data bits
Stop bits:	1

Actual data rate will be dictated by the host. The MCA will adjust itself to match the rate of the host computer.

B. API ERROR CODES

B.1. MCA Error Codes

These error codes are a subset of the InSpector MCA error codes. These codes will be returned to the application program. MCA error codes that are not on this list should not be encountered during normal execution of the InSpector API. If they occur, the API returns an application error code of F1. Possible error conditions returned to the application are:

sInvalidArgument	FC	Invalid argument(s) received
sInvalidLength	FA	Invalid length specified in the command
sAcquisitionOnError	F9	(a) Cannot load spectral data with acquisition on (b) Attempting to start acquisition with acquisition already on
sAcquisitionOffError	F8	Attempting to turn off acquisition with acquisition already off
sPowerNotOn	F7	Power must be ON to honor this command
sWriteError	F6	Write error detected during write to EEPROM
sInvalidNumChars	F5	Invalid number of characters in command
sAcquisitionNotOn	F4	Acquisition must be ON to honor this command
sDACVerifyError	F3	Write/Read verify error while writing to hardware DACs
sHVNotOff	EB	High voltage must be OFF to honor command
sPowerModeNotOn	EA	Power mode must be ON to turn high voltage ON
sGenSysError	F2	General System Error (usually unrecoverable)

B.2. Other Error Codes

sRCLineError	F1	Communication line error
sCOMMPortConflict	F0	Communication with another InSpector in progress
sMaxTimeReached	E9	Maximum Time Reached for Command
sAllDataNotReceived	E7	All Data not Received
sTransmitTimeOut	E8	Transmit Time Out

C. POWER SUPPLY ERROR CODES

These are error codes reported by the sGetPSStatus command. In most cases these represent a hardware-related problem in the Synchronous Communication Channel to the power supply.

01 Character timeout.

02 Unable to buffer SPI command(s)

03 Invalid RY character received after having sent SC

04 Unrecoverable Synch. Comm. Channel system error.

05 Invalid SC character

06 System error. SPIF flag not set at end of SPI transfer

07 SPI communication buffer full and pacing timer is OFF

08 Mode or Write-collision error

09 Invalid RY character during wait

0A Timeout while a SPI transfer was in progress

10 Not-ready when attempting to transmit

11 Not-ready when attempting to receive

12 Unable to communicate over SPI channel

13 Mismatch in received data after having sent B0

14 Mismatch in received data after having sent B1

15 Mismatch in received data after having sent EC

16 Mismatch in received data during receive of SR

17 Mismatch in received data during receive of ER

FC Receiver collision or noise error in the power supply

FD Invalid EC code received by the power supply

FE Invalid CC code received by the power supply

FF Invalid SC code received by the power supply

D. PROGRAM CONTENTS

DOSINSP.H	C function prototypes and constant definitions.
CITYPES.H	Type definitions used in DOSINSP.H.
DOSINSP.LIB	Static library of all functions used in the application interface.
CRTINSP.LIB	Static library of C run-time routines used by the application interface and not provided by BASIC compilers.
DOSINSP.BAS	Similar in content to DOSINSP.H but for use with BASIC for DOS.
DEMO.BAS	An example use of the interface written in BASIC for DOS.
DEMOC.C	An example use of the interface written in C.

WININSP.DLL	Windows 3.1 dynamic link library (DLL) of all functions used in the application interface.
WININSP.LIB	An import library of the DLL functions in the application interface.
WININSP.BAS	Similar in content to DOSINSP.H but for use with BASIC for Windows.

WARRANTY

This warranty covers Canberra hardware and software shipped to customers within the United States. For hardware and software shipped outside the United States, a similar warranty is provided by Canberra's local representative.

DOMESTIC WARRANTY

Equipment manufactured by Canberra's Instruments Division, Detector Products Division, and Nuclear Systems Division is warranted against defects in materials and workmanship for one year from the date of shipment.

Canberra warrants proper operation of its software only when used with software and hardware supplied by Canberra and warrants software media to be free from defects for 90 days from the date of shipment.

If defects are discovered within 30 days of the time you receive your order, Canberra will pay transportation costs both ways. After the first 30 days, you will have to pay the transportation costs.

This is the only warranty provided by Canberra; there are no other warranties, expressed or implied. All warranties of merchantability and fitness for an intended purpose are excluded. Canberra shall have no liability for any special, indirect or consequential damages caused by failure of any equipment manufactured by Canberra.

EXCLUSIONS

This warranty does not cover equipment which has been modified without Canberra's written permission or which has been subjected to unusual physical or electrical stress as determined by Canberra's Service Personnel.

Canberra is under no obligation to provide warranty service if adjustment or repair is required because of damage caused by other than ordinary use or if the equipment is serviced or repaired, or if an attempt is made to service or repair the equipment, by other than Canberra personnel without the prior approval of Canberra.

This warranty does not cover detector damage caused by abuse, neutrons, or heavy charged particles.

SHIPPING DAMAGE

Examine shipments carefully when you receive them for evidence of damage caused in transit. If damage is found, notify Canberra and the carrier immediately. Keep all packages, materials and documents, including your freight bill, invoice and packing list. Although Canberra is not responsible for damage sustained in transit, we will be glad to help you in processing your claim.

OUT OF WARRANTY REPAIRS

Any Canberra equipment which is no longer covered by warranty may be returned to Canberra freight prepaid for repair. After the equipment is repaired, it will pass through our normal pre-shipment checkout procedure.

RETURNING EQUIPMENT

Before returning equipment for repair you must contact your Regional Service Center or one of our factories for instructions. For detector repair, contact the Canberra Detector Division in our Meriden, Connecticut, factory for instructions. If you are going to return the equipment to the factory, you must call first to get an Authorized Return Number (ARN).

When you call us, we will be glad to suggest the best way for you to ship the equipment and will expedite the shipment in case it is delayed or lost in transit. Giving you shipping advice does not make us responsible for the equipment while it is in transit.

SOFTWARE LICENSE

You have purchased the license to use Canberra software, not the software itself. Since title to this software remains with Canberra, you may not sell or transfer this software. This license allows you to use this software on only one compatible computer at a time. You must get Canberra's written permission for any exception to this license.

BACKUP COPIES

Canberra's software is protected by United States Copyright Law and by International Copyright Treaties. You have Canberra's express permission to make one archival copy of this software for back-up protection. You may not copy Canberra software or any part of it for any other purpose.